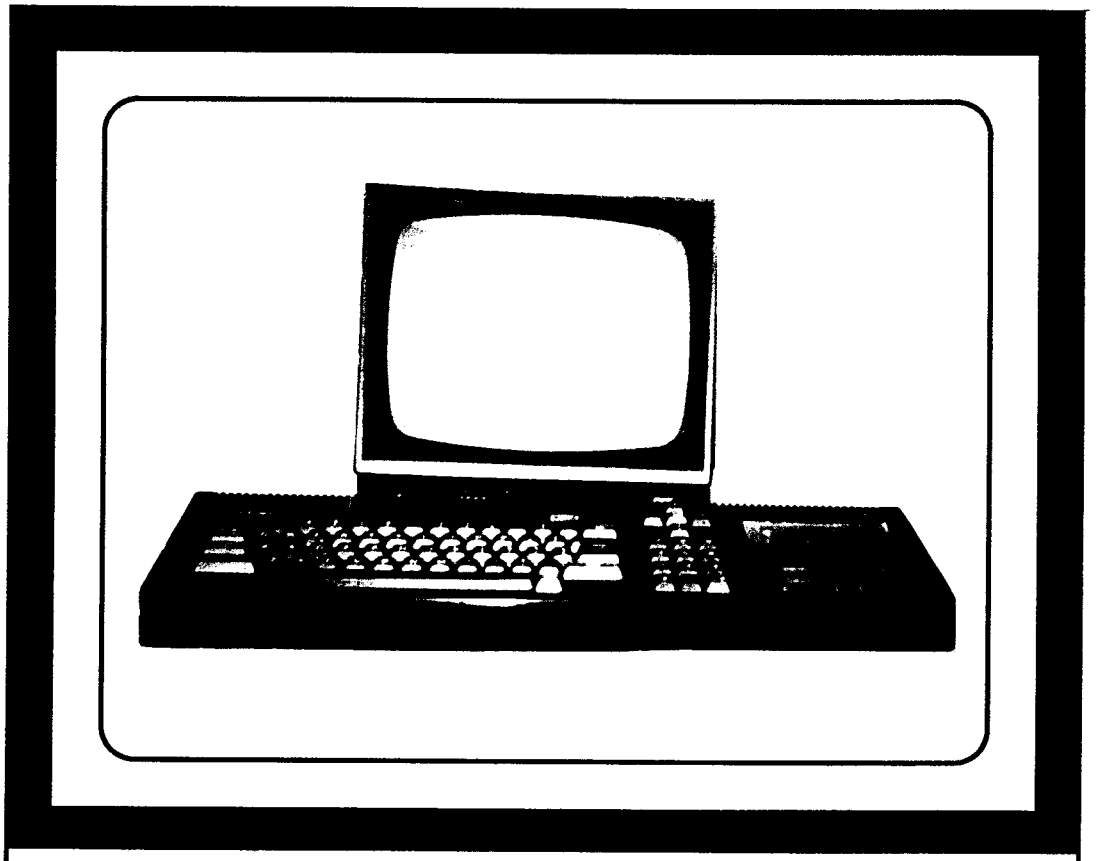
Price £1.10

PRINT-OUT

ISSUE EIGHT



Written by Thomas Defoe and Mark Gearing
Technical Contributor: Bob Taylor

INCLUDING:

INDEX

Miscellaneous

Features

Reviews

Programming

We would like to express our thanks to Mr. Gearing and Black Horse Agencies Januarys for the continued use of their photocopier in producing Print-Out.

Editorial

WELCOME TO ISSUE EIGHT OF PRINT-OUT !!!

Many thanks to all of you who sent in Christmas cards and presents - they were appreciated very much. We hope that you'll find this issue worth the wait, and we'd once again like to apologise for the delay.

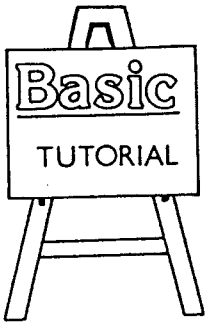
As you can probably see, some of the artwork in this magazine (along with some titles) has been produced on an Apple Macintosh & Laser Printer. Over the next few issues we intend to design as many of the titles as possible in this way, due to the far higher quality of printing. However, don't worry - we're going to continue using our trusty CPC for the main body of Print-Out !!

We've also been making changes to the magazine, based on the results of our questionnaire. It's been decided that a 'Readers' Letters Page' and 'Technical Tips' columns will be re-introduced. Also from next issue, we hope to have a regular CPM section.

If you've any queries or problems with the CPC, please write to us at the address below & we'll do our best to solve your problem. We guarantee that all letters will be answered personally by one of the writers of Print-Out. The address is the same for all orders, and is printed below:

PRINT-OUT, 8 Maze Green Road,
Bishop's Stortford, Herts CM23 2PJ.

COMPETITION WINNERS...
PAGE 8



BEGINNER'S BASIC

PART EIGHT

This issue, we begin our exploration of some of the more advanced concepts in BASIC programming. So we will start with an area which a few of our readers have requested - that is DATA statements and their uses.

The best way of illustrating the main benefits of using data is to give a few example programs. However before we can do that, we need to have a brief look at how the necessary commands work.

A data statement is basically a means of storing information until such time as we need to use it. It always takes the form of DATA followed by a list of the bits of information which we wish to store.

eg. DATA January,February,March,April

Every of piece of information is called an 'item' and all items are separated by a comma. Whenever we need to retrieve this stored data, we must use the READ command. The correct syntax is as follows: READ plus a variable (eg READ month$) and whatever item is read will be placed in this variable.

READ always looks at the first bit of data that it finds in the program, and then gets all of the information up until the comma (or the end of the line) and puts it in the variable which follows the READ command.

If another READ command is then executed, it looks at the next piece of data, after the one it has already got. Try this example:

```
10 READ a$:PRINT a$
20 READ b$:PRINT b$
30 READ c$:PRINT c$
40 DATA January,February,March
```

When this is RUN, you should get the three months printed on the screen. Now suppose you change the program to:

```
10 FOR i=1 TO 3
20 READ a$:PRINT a$
30 NEXT i
40 DATA January,February,March
```

You should get exactly the same result. All it does is make it a bit neater & more adaptable. If you follow it through; Lines 10 and 30 set up a loop which is executed three times; every time Line 20 is executed, it READs the next piece of data (which is contained in Line 40). Thus the first time round it looks for the first bit of data in the program (ie January); the second time it finds the next piece of data (ie February); and on the last time, it gets March.

Suppose you now add the following line and re-RUN the program:

```
50 DATA April,May,June
```

Nothing new will happen and this is because we forgot to tell the computer that it must now READ six bits of information. To do this, just change Line 10 to:

```
10 FOR i=1 TO 6
```

What happens if you ask the compute to read more information than you've really included? Try it and see (change it to 10 FOR i=1 TO 7) and the poor CPC gives up and you'll get a rather unpleasant 'DATA exhausted' error!!

The great beauty of using DATA statements lies in your ability to change the stored information very easily. Numbers, letters, words or a mixture of them may be stored in a DATA line. So, change lines 20, 40 and 50 to:

```
20 READ a$:READ b:PRINT a$;"    ";b
40 DATA January,2,February,6.54,March,3
50 DATA April,4,May,0.45,June,9.21
```

Now, you'll get a little table of figures for each of the six months. Line 20 had to be changed in order that the numbers were read into a numerical variable and the text into a string variable. Line 20 could also have been re-written as:

```
20 READ a$,b:PRINT a$;"    ";b
```

Indeed you can read as many variables as you like with just one READ command. It is also possible to have as much information in a single DATA line (or statement) as you like, providing it is not more than 255 - the maximum length of any BASIC program line.

There is one other important trick which can be used in DATA statements, and that is if we wish to include a comma in our data. However, usually a comma is used in indicate where one piece of data ends, and another begins. Thus the line 60 DATA WELL. IT'S NICE TO SEE YOU! would be treated as two separate pieces of information when they are read (ie. WELL and IT'S NICE TO SEE YOU!). Still, it's possible to overcome this by storing the text in inverted commas. Hence,

```
60 DATA "WELL. IT'S NICE TO SEE YOU!"
```

would be treated as just one piece of information when it is read. This be very useful in things like adventure games where a certain command produces a certain response. However, the real power of DATA statements hasn't been introduced yet! All of the extra possibilities exist because of an associated command - namely RESTORE plus a line number.

Before looking at the RESTORE command, it is worth bearing in mind that data can be stored anywhere in the program - at the beginning, in the middle, or all jumbled up (the computer will still follow it all through logically, reading the first bit of information it comes to, and then carrying on from there). It makes more sense, however, to put all of the data in one place, so that we can find it and alter it easily.

It is extremely important to make sure that all of the DATA is right and that we haven't missed any bit out, as it's notoriously difficult to track down where there is an error.

As mentioned elsewhere in this article, the READ command looks for the first bit of data that it comes across in the program and then uses that. However, our computer is provided with a means of altering where the first bit of information to read is - this is the RESTORE command which has many uses.

The RESTORE instruction is not entirely precise as it can only specify a line number as the start location. Suppose, we were writing a program which needed us to print out a list of months and some figures twice. Without using RESTORE, we would need to write out the data twice (once for each time it was needed). This is needed because it would print out the information once and, in order to print it out a second time, need to be supplied with the data again - as it otherwise it would run out of data and give us a syntax error.

Using RESTORE, the program below, which is a modification of the others, will allow you to print out the data twice with no difficulty.

```
 5 FOR j=1 TO 2
10 FOR i=1 TO 6
20 READ a$:READ b:PRINT a$;"    ";b
30 NEXT i
35 RESTORE
36 NEXT j
40 DATA January,2,February,6.54,March,3
50 DATA April,4,May,0.45,June,9.21
```

Here, RESTORE is used without a line number, and in this case it merely tells the computer to look for the data as if it were starting again from scratch (ie. it would then start reading from line 40 again).

Finally here's a quick example of how RESTORE can be used to set up variables with different values depending on the user's response.

```
10 INPUT "Do you want the months in French or English (F or E) ? ",lang$
20 IF UPPER$(lang$)="F" THEN RESTORE 60 ELSE RESTORE 70
30 FOR i=1 TO 6
40 READ month$:PRINT month$
50 NEXT i
60 DATA janvier,fevrier,mars,avril,mai,juni
70 DATA January,February,March,April,May,June
```

If you have any comments on anything to do with computing, then please write to:

PRINT-OUT, 8 Maze Green Road,
Bishop's Stortford, Hertfordshire CM23 2PJ.

DISC DRIVE PROBLEMS

I have seen second-hand disc drives from machines such as the BBC in my local computer shop and I would like to know whether these can be used on the CPC with the appropriate DOS software.If so, articles on installation of non-CPC-specific peripherals would be much appreciated.

TONY SHEPPARD, BRIGHTON

PRINT-OUT: As long as the disc drive complies to the Shugart standard it should be possible to use it on the CPC. However, it is not something that should be undertaken by the faint hearted. The best place for information on installing a disc drive is in the Firmware Manual (Appendix 12.9/12.10) where it gives a list of the pin-outs and some other important notes. Providing it is a second drive, you should not need an alternative DOS to use the drive, in which case it will act just like a standard Amstrad drive (you may want to use something like RAMDOS to access the extra size of the disc). If you do not already have an Amstrad first drive, you'll need to buy one as it includes the basic DOS.

AMSTRAD WRITES.....

We recently wrote to Amstrad with some of the numerous questions which you have raised concerning the Plus computers and, within a week, we'd received a letter in reply. As these queries will be of interest to many readers, we have decided to print the questions and their replies below. Amstrad also promised to answer any further questions which our readers might have. So get writing!!

Do you plan to make a piece of hardware which will allow existing CPC owners to run cartridge software ?

Amstrad do not have any plans at present for such a widget.

Will 6128+ owners be able to load tapes ?

No, CPC 6128+ owners will not be able to load tapes.

Will CF-2 discs remain at the same price ?

The CF-2 discs will not be reduced in price.

Will the 464+ be able to use a disc drive ?

As yet there are no disc drives available for the CPC 464+.

Are cartridges likely to be reduced in price ?

No, CPC cartridge software is to remain at the fixed price of £30.

Will the new hardware be accessible to the user ?

Regretfully the user will not be able to program the hardware himself.

Are the Pluses compatible with all CPC hardware ?

The computers will not necessarily be compatible with existing hardware.

QUESTIONNAIRE RESULTS

The response to our questionnaire was excellent and as a result we will try to produce a magazine which is in tune to its readers' wishes. Printed below are the results of some of the more important sections of the questionnaire.

BASIC Programming	MORE - 40%	**SAME - 60%**	LESS - 0%
M/Code Programming	MORE - 40%	**SAME - 50%**	LESS - 10%
Homebrew Reviews	MORE - 35%	**SAME - 50%**	LESS - 15%
Public Domain	MORE - 30%	**SAME - 50%**	LESS - 20%
Using CPM	**MORE - 75%**	SAME - 10%	LESS - 15%
Solutions to problems	**MORE - 75%**	SAME - 25%	LESS - 0%
News and General	MORE - 35%	**SAME - 55%**	LESS - 10%
Type-in programs	MORE - 40%	**SAME - 50%**	LESS - 10%

And as a result of this, our letter page has been reintroduced as has a section which is dedicated to solving people's problems. But as these areas are totally dependent on readers' letters, we need you to keep writing in with comments and queries. Also, as from next issue, there will be a regular CPM section, dealing with every aspect of both CPM 2.2 and CPM Plus.

The winner of the PRIZE QUESTIONNAIRE was M KALIBABKA from SALTBURN-BY-SEA and will receive a copy of PROTEXT on tape.

WINNERS

We also have a large number of winners from the competitions which we ran in Issues Six and Seven and they are all printed below.

To start, well done to SUSAN ILSLEY from PRESTBURY and OWEN BROWN from TOTTON who have both won copies of all of the programs listed in Print-Out on disc.

In Issue Seven we ran a competition in celebration of our first birthday, and we had a tremendous response. Anyway, the three lucky people who each win a copy of the fabulous Tearaway program from CPC Network are:

ROB MUNDIN from WATERLOOVILLE

A J COOK from WORTHING

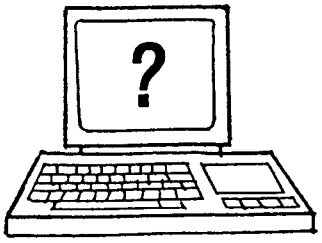
S J LEE from HUNTINGDON

The other winners are D WEBB from CHORLEY who wins a set of dust covers and also KEITH HANKIN who should soon be receiving a copy of Rick Dangerous II on tape.

Congratulations to you all.

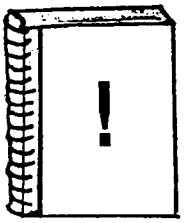
HELPLINE

Every month we seem to get so many queries regarding various areas of computing on the CPC, that we've decided to try and start up a helpline section. However, the success of this area of Print-Out largely depends on readers contributions. Therefore, if you wish to be entered into a regularly updated list, either with an offer of help or a query, please get in touch with us at the usual address.



TECHNICAL TIPS

by Bob Taylor



S. Messina of Heywood has written in again enquiring about several matters. Firstly he would like a faster way to write machine code. I'm not too sure just what he means. If it's the writing process he'd like speeded up, I would suggest that he starts a library of routines and uses these as building blocks in future programs. Alternatively if programs were written in BASIC & these were converted into code using a BASIC compiler, writing machine code could be dispensed with altogether (what a terrible thought!).

On the other hand, if the intention is to try and write programs which run faster, then compiled routines are notoriously longer than custom written code, needing to be capable of handling every possible BASIC diversity. Even a library routine should be pared down to run faster for a particular situation.

Writing faster routines isn't something to which other magazines and authors address themselves. In PRINT-OUT, we do give tips on the subject, especially via the comments in routines for assembly and these should be studied. In fact it is a good plan to always look at how other people have written programs - what may then be seen will be not only how to write code better, but alas, in some cases, how not to write it: don't be afraid to reject such programming.

Although tips on faster code are a great help, the real savings in time are to be made in the way that the routine does its job (its algorithm). The choice between algorithms is limited if the programmer is unaware of the existence of alternatives and this is where studying other people's code comes in. Also, the ability to look at the job to be done from a different angle is a powerful tool for the programmer to exercise. Add to that a measure of anarchy - do not just accept any rules you have been taught, but try to see if there is a way around any limitations (I hope that my articles on RSXs illustrate this).

There is no definitive version of a routine: it can be rewritten a number of different ways, and amongst these may be one that runs faster. Once the routine is written and runs Ok, don't just leave it at that, but look at it again to see if there are points which can be improved upon. Good programming!

Another of the things which Mr Messina was interested in was a program to check how many 'T' states a machine code instruction takes (we're still on the theme of faster routines, I think). A 'T' state is a microprocessor clock pulse and it takes a certain number of these for an instruction to operate.

The only timing device in the CPC that is available to the programmer is the interrupt clock which has a frequency of 300 ticks/Second. This is much too long a period between ticks to time instructions which could take only a microsecond, so the trick is to repeat the instruction many times and divide down the resulting time.

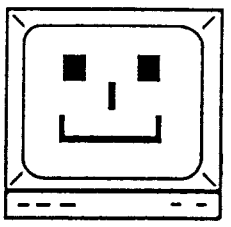
In the BASIC program below, the maths in line 50 gives a reasonably accurate representation of the number of 'T' states an instruction takes, but in doing so it illustrates a problem which any program designed for this purpose must face. In the first article on RSXs in Issue Five, I explained that the microprocessor clock ran at 4 million pulses/sec but this was virtually slowed down to approximately 3 million by the insertion of wait states - clock pulses when the microprocessor does nothing. (The reason for this slowing is to limit memory access to one per 1uS). If we chose to time an instruction that does not need to access memory, the result given by the program stays accurate; if it does use memory however, or even if there is more than one byte to the instruction, the result becomes increased by the number of wait states inserted.

To use the program, enter the op code(s) of the instruction in a data statement in a new line 100. For example enter: **100 DATA 78**
This is the code for the instruction LD A,B. When run the program will print out the result, 4 T states, six times and this is the correct result for such an instruction. Now enter: **100 DATA 7e** this is LD A,(HL)
The result now is 8 which is not correct (the right figure is 7 T states) and so the extra one is a wait state. Sometimes the six results may differ: the reason for this is probably that a system interrupt occurred during the timing and lengthened one result slightly.

Because of the possible discrepancies the program is provided for interest only. A full list of every type of instruction with its correct timing (including the illegal instructions) are available from us if you send an SAE.

Not all instructions can be used in the program as many will cause a crash. Of these a few can be used in combination with others to get round this problem:
100 DATA e5,e1 this the code for PUSH HL:POP HL
Do not use any instructions which alter the B or C registers (these are used in the program itself), and only alter memory with great care.

```
[71] 1 'Machine Code Timer by Bob Taylor (copyright 1990)
[BB] 10 ON ERROR GOTO 70
[BD] 20 FOR a=&8000 TO &8100:READ b$:POKE a,VAL("&"+b$):NEXT
[A3] 30 FOR a=1 TO 6:t=TIME
[26] 40 FOR n=1 TO 100:CALL &8000:NEXT
[F4] 50 PRINT INT((TIME-t-184)/8.2+0.5)"T":NEXT
[E8] 60 END
[98] 70 IF ERR=4 AND ERL=20 THEN RESUME 30
[23] 80 ON ERROR GOTO 0:RESUME NEXT
[77] 90 DATA 01,e8,04
[3D] 110 DATA 0d,c2,03,80,05,c2,03,80,c9
```



All programs in Print-Out have Linecheck codes which are enclosed in brackets at the start of a line. Don't enter them in as they're designed to be used with Linechecker to eliminate errors when typing in programs which appear in this magazine. Please note, all programs will run whether Linechecker is being used or not. For information on how to use Linechecker, please see Issue Three.

PRINTER DIP SWITCHES

We have received a query from Mr D Sherwood regarding the overriding of the DMP 2000 Printer's DIP switches by the use of software (actually by sending ESC code sequences to the Printer in most cases). As far as I can ascertain only the following switches can be bypassed in this way:

DS1-1,2,3	'ESC',"R",n			(chapter 6, page 4)
DS1-4	JUST SEND AN EXTRA LF			
DS1-5	'ESC',"9"	or	'ESC',"8"	(chapter 6, page 3)
DS1-6	'ESC',"C",0,11	or	'ESC',"C",0,12	(chapter 4, page 4)
DS1-7	* NOT POSSIBLE *			
DS1-8	'ESC',"x",1	or	'ESC',"x",0	(chapter 3, page 4)
DS2-1	* NOT POSSIBLE *			
DS2-2	'ESC',"O"	or	'ESC',"N",n	(chapter 4, page 4)
DS2-3,4	CHAPTER 5 WITH CHAPTER 6, PAGE 5 ONWARDS			
DS2-5	* NOT POSSIBLE *			
DS2-6	* NOT POSSIBLE *			
DS2-7,8	CHAPTER 3, PAGE 4 ONWARDS			
DS2-9,10	* DO NOT USE *			

A quick peek inside my DMP 2000 shows that DS2-9 and 10 are both connected into circuit. Why we shouldn't use them I don't know and I'm not willing to sacrifice my printer to find out. Perhaps a reader already knows and can enlighten us!

BINARY FILES

```
10 REM File Header Reader (c) 1991
20 DATA 21,92,00,46,21,99,00,11,00
30 DATA C0,CD,77,BC,ED,43,93,00,ED
40 DATA 53,95,00,2A,95,00,CD,83,BC
50 DATA 22,97,00,CD,7A,BC,C9,00,00
60 RESTORE 20
70 FOR i=1 TO 36
80 READ a$:POKE &6F+i,VAL("&"+a$)
100 NEXT i
120 INPUT "Enter filename: ",file$
130 b=LEN(file$)
140 FOR i=1 TO b
150 b$=UPPER$(MID$((file$),i,1))
160 POKE &98+i,ASC(b$)
170 NEXT i
180 POKE &92,b
190 KEY 138,"PRINT HEX$(PEEK(&95)+
    PEEK(&96)*256),HEX$(PEEK(&93)+PEEK
    (&94)*256),HEX$(PEEK(&97)+PEEK(&98)
    *256)"
200 CALL &70
```

I have a game which is included on a tape which had a number of other games and programs and was always a bit of a chore to find, plus the length of time it takes to load from tape. As I own a disc drive I thought it a good idea to transfer this particular game to disc for speed of loading. The main program which isn't protected loaded from tape. There are three binary files but after many attempts I cannot transfer these. Despite following the instructions in The manual for the DD1. I keep getting a 'Memory Full' message.

JIM PROCTOR

If you run the program on the left and follow the instructions, then when you are returned to BASIC, press F. or '.' on the keypad. The first number is the START ADDRESS, the second the LENGTH & the last is its EXECUTION ADDRESS.

If the first file has a name of FILE1; start of &4000; length of &2000 and an execution address of &2678, you would use the following lines to copy it:

```
|TAPE:MEMORY &4000-1:LOAD "FILE1"
|DISC:SAVE "FILE1",B,&4000,&2000,&2678
```

CONTROL CODES IN LISTINGS

Mr D Webb of Doncaster collects BASIC programs and has noticed that Control Code characters are sometimes present within quotes instead of in the usual form CHR$(n), and wonders why programmers have done this.

Whenever I've used this form of entry it's because the characters take a few less key strokes to enter compared to the conventional way, with the added bonus of taking less room in memory. A further plus is that it does runs fractionally faster. However, this method does make a program more difficult to check, unless the programmer is able to recognise the symbols quickly through practice. Using Control Codes in strings is not a practice to recommend for listings which other people need to copy.

Control Code characters can be entered into strings in a program by pressing CONTROL together with the keys A to Z, [\ ] ^ _ . In this form, or in the more usual CHR$(n) way, they can be used for various screen controlling operations as detailed in the handbook. Note that some require parameters, which can also be entered as symbols or characters (whose ASCIIs are the values required). It is unfortunate that one of the most useful values, CHR$(0) ie the NUL character, is not available in this form.

When LISTing via a Printer, no Control Code characters will be printed: most will be ignored, while the rest will cause the Printer to operate in a different way, giving unexpected results. Control Codes can be displayed using the correct ESCAPE sequence (ESC, "I", 1 for the DMP 2000) but of course this cannot be used from within a LISTing.

Graphics characters are also present in some programs. One very good reason for doing this is to obtain an early representation (within the program listing) of what will appear when RUN. Graphics characters can only be used in this way by first printing separately on the screen using the command PRINT CHR$(n) and then COPYing the result into the program line. One Graphics character that can't be COPYed is CHR$(128) which prints on the screen as a Space - COPYing this will result in a SPACE, and not the proper character.

Sometimes through accidental faulty memory management, the UDG (User Defined Graphics) area becomes corrupted by other data. If this should happen, any characters stored there will display incorrectly, because they are made up of the corrupted bytes. Again, these 'false' characters will COPY correctly.

When it comes to Printing a LISTing of any Graphics characters, we run into several problems. Since the CPCs can only send 7 bits to a Printer, and Graphics characters are usually CHR$(128) and above, the eighth bit which they need, will be lost. Should we have an 8 bit Printer Port, bit 7 will be sent but the result will depend on the Printer. The DMP 2000 for instance produces italics: either a jumbled sequence of characters for codes 128 to 159, or characters 32 to 127 for the rest. A few other Printers do hold graphics characters above code 127 but it is extremely unlikely that these will match those present in the CPC.

HOMEBREW SOFTWARE

MAGICIAN'S APPRENTICE by SIMON AVERY

Following on from the highly commendable adventure games, FIRESTONE and JASON AND THE ARGONAUTS is MAGICIAN'S APPRENTICE.

This game has the same style as the previous ones, but this is not surprising as all three have been designed using the Quill.

In it, you play the role of Wuntvor, apprentice to the magician Ebeneezum who one day disappears. Desperate with worry, you take it upon yourself to find and rescue him from whatever perils have befallen him.

There was no loading screen which I found a little disappointing as I feel it is important to have graphics in an adventure game of this type, to break up the monotony of the text.

The locations in the game were fairly interesting and varied although some of the descriptions were rather standard to most adventure games. The places that I visited were nothing out of the ordinary - they had all been done before. Having said this, I did get enjoyment from some of the puzzles that Simon had invented, as they ranged from the very easy to the extremely difficult.

The game lacked any truly original or novel ideas. As usual in Simon's games, there were a few characters who I could talk to, and this helped to liven things up a bit. A score feature was incorporated into the game which I liked and again Simon's own brand of humour was very apparent.

Whilst humour can sometimes spoil the atmosphere of a game like this, all of Simon's adventures contain various jokes and they're used to good effect. Indeed they add to the game greatly - some of them are even funny!!!

Unfortunately, there were no graphics with this game which was again a little disappointing - sometimes, though, it can be a blessing as many homebrew authors include them for no real reason and many drawings are indecipherable. Therefore, the lack of graphics isn't a real problem in this game, especially when the text wasn't boring.

The atmosphere created in the game could never really be described as 'tense' or 'exciting' but the adventure did paint a clear and vivid picture in my mind, and, when coupled to his 'strange' sense of humour, the game could become quite addictive. A failing of most homebrew adventures is that there's no real feeling of threat or danger - and this was no exception.

However, setting aside these minor niggles, I enjoyed it for a while, but it wasn't very different from the vast majority of adventure and it became a little tedious for me, especially after having played so many other homebrew games. But apart from that, the game was good and well polished.

MAGICIAN'S APPRENTICE costs £2.50 on tape and £2 if you send a disc yourself. The game represents good value if you like adventure games, if you don't, then I wouldn't recommend it to you. However, it's a good effort, interesting at times, humorous, and a must for adventure gamesplayers.

Simon Avery has been a prolific adventure writer and listed below are some of his many adventures. Most are available on both tape and disc and are incredibly cheap, as they are in fact public domain. No doubt Simon will be glad to inform you of their prices and anything else that you wish to know about them.

The Public Domain adventures which are on sale are as follows:
'ROOG' 'TIZPAN' 'WELLADAY' 'FIRESTONE'
'JASON AND THE ARGONAUTS' 'SPACED OUT'
'DUNGEON' 'DOOMLORDS (Parts 1,2,3)'
'CAN I CHEAT DEATH ?' and 'ADULT 2'

'MAGICIAN'S APPRENTICE' is the only non-PD adventure and is reviewed above.

You can reach Simon at: MORDEN FARM, OLD EXETER ROAD, CHUDLEIGH, DEVON TQ13 0DR.

TAPE ADVENTURES

Another author of homebrew software, Tony Kingsmill, wrote in to tell us that he now has a collection of his best adventures available on tape. All of the games have been reviewed in Print-Out and they are REVENGE OF CHAOS, ISLAND OF CHAOS, ALIEN PLANET and LORDS OF MAGIC. Up until now, these have only been available on disc and were given favourable reviews by us. Tony has called this new selection 'THE TAPE ADVENTURE COLLECTION' and the pack is available from him at the price of £4.50 for all four games. You can contact Tony at the following address:

TONY KINGSMILL, 202 PARK STREET LANE, PARK STREET, ST ALBANS, HERTS AL2 2AQ.

STUNNER

In the Issue Six of Print-Out I reviewed three games written by Adrian Sill - Crimson, Traitor and Snookered. Now he has included these games on a compilation called 'STUNNER' which comprises eight games or tutorials.

The collection was presented in an extremely polished manner with an impressive title screen, complete with clear, bold text. After selecting an option from the main menu, detailed game instructions appeared on the screen and, along with the crisp and neatly printed accompanying documentation, the game's scenario and objectives were clearly detailed. Having eight games in the compilation, there's a wide enough choice to provide hours of enjoyment.

The first game was called ANIMAL GAME, which is a simple variation on the old 'animal, vegetable, mineral' theme. It was fairly straightforward to play, but I felt the game needed some finishing touches done to it, and I found its style a little too 'tacky'.

Next was CRIMSON, the arcade adventure I reviewed in Issue Six. This was the best game of the compilation and it involves steering a small spacecraft through a screen which is littered with various obstacles. Unfortunately your ship keeps on falling and you cannot control its speed, but only its height. It was a well produced and very polished game - its simplicity and originality made it hugely addictive. It's just a shame that it hasn't got even more levels as I certainly enjoyed playing it.

FRENCH TESTER was the next program. It is a multi-choice educational program which lists a word in either English or French and you have to choose the right translation from the four possibilities given. However, it was clearly aimed at those just beginning French and it could have been enhanced with different skill levels. Still, the program was nicely laid out and I liked the sprite that moved across the screen when you got it right or wrong. All in all a very useful, even if a bit limited.

MATHS TUTOR follows in the same vein as the last one. You could test yourself on multiplication, addition, subtraction, division or a mixture of all four. It was virtually identical to French Tester, except that numbers were involved and not words. Again it really needed various skill levels - but still good fun.

MAP UK was an interesting utility. It printed a map of the UK with a grid on, and allowed you to add borders or to place a number of known towns on it. Quite how useful this is I'm not sure, but it was certainly well done and detailed.

MR TRIVIA was probably the best educational game of the compilation. It was more interesting than the others and quite challenging, although not impossible. Again, you are asked twenty questions and have to choose between one of the four possible answers.

SNOOKERED and TRAITOR were the two last games on Stunner and these were both reviewed in Issue Six. These were also very good games.

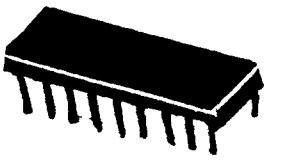
I thought that it was a good blend of arcade and educational games along with the odd utility. One or two of the programs, as I have mentioned, were not very good at all but the majority were excellent. Crimson, Traitor and Snookered were certainly the dominant games, easily being the best three.

At a price of £4.99 on tape and £6.99 on disc it is definitely a bargain well worth having. I'd hours of fun and enjoyment from it, and I'm sure you will too!

ADRIAN SILL. 19 SHERWOOD DRIVE. 5 LANE ENDS, SKELLOW, DONCASTER, YORKS DN6 8NY.

Machine ....Code....

Programming your Z80



COMMANDS

Welcome to this issue's article on Machine Code. As from Issue Nine we will be putting the commands that we have come across in programs and looking at some Machine Code routines. However, this issue we're going to be investigating a few more commands - some of which are new and some which we've already used without explaining fully.

NEG

The first of these, is the NEG mnemonic. This only works on the Accumulator and it makes the number in A negative. Hence at the end of the example below the A register holds -105. Of course, this is stored in 'Two's Complement' form (see Issue Seven for more details) and so will be represented by 10010111 in binary - 151 in decimal.

```
ORG &8000
LD A,105        ; A holds 105
NEG             ; A is now negated (holds -105)
RET
```

DJNZ

The next one is very useful and we have come across it briefly before - the DJNZ instruction. This command is often used in looping and it relies on some of the condional techniques which we looked at in a previous issue. DJNZ is really an abbreviation of a couple of commands (ie it's a short cut) and these are:

```
DEC B
LD A,B
CP 0
JP NZ,label
```

You can see why we have an abbreviated form! What happens is as follows: At the start of your loop, you load the B register with a value which is the number of times that you wish the loop to be executed. This LD instruction is followed by a label (such as '.loop1') and then the part of the routine which you want to be used every time the program goes around the loop. At the end of the loop, you put a DJNZ instruction (in our case this would be 'DJNZ loop1'). What this means is that each time the computer goes round the loop, it decrements B by 1 and it looks to see if B equals zero. If it does, the computer continues with the rest of the program. If it doesn't, the CPC jumps back to the beginning of the loop & continues executing the routine from there.

Of course, it's important to ensure that the B register is not corrupted in the routine which is enclosed by the loop. To prevent this we can use PUSH & POP instructions (which we looked at in our last issue) to preserve the B register. It is, therefore, also important to make sure that anything else that is PUSHed onto the stack is also POPped off (often we can ignore this problem by ensuring that we don't use B in our routine).

It is sometimes useful to be able to alter the value of loops, according to the result of a certain action within the loop. This can be achieved by altering the value of B during the actual loop.

The routine below illustrates these points with a fairly pointless example. It goes round the loop and prints all the numbers from one to nine. Each time it also asks if you want to print a number again (it prints a question mark & waits for you to enter something). If you want it to repeat a number, then enter 'Y' & the program will automatically adjust the size of the loop. Otherwise, any other key will be treated as if you had typed 'N'.

```
ORG &8000          ; tells the computer where to put the routine in memory
   LD B,9          ; B is loaded with 9
   LD A,49         ; A is loaded with 49 - this is the ASCII for '1'
.loop
   PUSH BC         ; Put BC on the stack to preserve register B
   CALL &BB5A      ; Print the character which is in A
   INC A           ; A is increased by one (A=A+1)
   PUSH AF         ; The new value of A is put on the stack to preserve it
   LD A,63         ; A is loaded with 63 - this is the ASCII for '?'
   CALL &BB5A      ;  and the question mark is now printed
   CALL &BB06      ; The computer waits for a key to be pressed
   CP 89           ;  and the value returned is checked against 89 ('Y')
   JP Z.dummy1     ;  and if it is 'Y' it jumps to the label 'dummy1'
   CP 121          ;  otherwise the value is checked again 121 ('y')
   JP NZ.dummy     ;  and if it is not 'y' it jumps to the label 'dummy'
.dummy1
   POP AF          ; AF is removed, A contains the next number to print
   POP BC          ; BC is removed, B contains the number of times to loop
   INC B           ; B=B+1, the number of loops is increased by one
   DEC A           ; A=A-1, the number to print is decreased by one
   PUSH BC         ; BC is put back on the stack
   PUSH AF         ; AF is put back on the stack
.dummy
   POP AF          ; AF is popped from the stack
   POP BC          ; BC is popped from the stack, B holds number of loops
   DJNZ loop       ; B is decreased by one and checked to see if it equals
                   ;  one and if it doesn't the program jumps back to the
                   ;  label 'loop' and the loop is repeated
   RET             ;  else the program is ended
```

The above program can seem quite complex at first glance, but it uses only commands and concepts which we have already talked about. The best way of understanding it is to follow it through, in the order the computer would, with real numbers in and see if you can make sense of it. The routine has a branching part contained within another conditional loop and this can add to its complexity. As from next issu we will be starting to look at more useful routines and trying to use the commands, which we have looked at, for some worthwhile purpose.

DIRECTIVES

The final set of mnemonics that we're going to look at aren't real commands but are what are known as directives. These are not machine command instructions but are special notes for an assembler, and they make life very much simpler for the machine code programmers.

Most assemblers have a large number of directives, but we are only going to look at some of the more common ones and they are: DB (or DEFB or DEFM or BYTE), DS (or DEFS or RMEM), DW (or DEFW or WORD) and ORG.

Note that for some assemblers there may be other alternatives to these.

ORG has been used in almost every Machine Code program that we have written, and tells the computer where that particular routine is to be located in memory. For example, the above program begins with ORG &8000 and it tells us that it is to be located at address &8000.

DB (is an abbreviation for Define Byte) and does exactly that. The DB directive tells the computer to put one byte of data aside, and then to set that byte to whatever the value following it is. For example DB 45 would set that byte to have a value of 45, and so is very useful for storing data. You can also store text in this way, or a mixture (eg DB "Hello out there",13,10) However note that some assemblers might insist that you use DEFM if you want to hold any text.

DW (is an abbreviation for Define Word). In code, a WORD is a 16-bit number (eg &C000 or &0567) and, as we already know, a 16-bit number needs two bytes to store it. Also, the number is held with its low byte first, followed by its high byte (eg &67,&50). What the DW directive does, is to sort all of this out for us automatically - it reserves the necessary memory and then stores the number with its low byte first, then its high byte. So an example of it would be DW &0567.

DS is possibly the least used of all of the directives and merely reserves some space (upto 256 bytes) which can be used as a buffer. All of the bytes that have been reserved is then filled with zeros, ready for use as as a storage place.

BASIC

Poking Around

M/CODE

A Selection of Useful Tipr

This month's 'Poking Around' is mainly concerned with the peripherals which can be added onto the back of your trusty CPC. There are also some handy tips to use in your BASIC programs and the usual collection of handy PEEKs and POKEs.

Detecting a Multiface

First of all, here's a neat way to test for a Multiface device on a CPC. If it finds a Multiface, it will crash the computer. You can fool it by turning the Multiface off, as explained in your manual. Just type `OUT &FEE8,&E8`

Using CTRL-ENTER

It's annoying for 6128 users to be unable to use the CTRL-ENTER combination to some useful effect. On the 464 with cassettes, it's fine to be presented with RUN" followed by ENTER automatically. However on a 6128 or 464 with a disc drive that just gives an error message - the ROM really should have been rewritten for discs! Anyway, here is a little tip that will give RUN" without the ENTER, ready for you to enter the required filename.

For a 6128: `POKE &B5AD,0` For a 464 with a disc drive: `POKE &B463,0`

On-line Printer ?

Another useful tip for checking the state of your peripherals, is this one which tells you whether your printer is switched on and is 'On Line', by use of the command `PRINT INP(&F500)`. If it is ON and is 'On Line' this should give you the value &1E (30) and if it is OFF it will give 94 (&5E). Unfortunately, there is no way of detecting the 'Paper Out' sensor by software.

BASIC tricks

Now for a couple of BASIC programming techniques. Sometimes it's necessary to increase a variable each time something is done. If this value should revert to zero when it reaches a certain number, this is how it is usually done:

```
110 a=a+1:IF a=10 THEN a=0
```

Unfortunately, this has the side effect of meaning that any common actions must be on another line - which is sometimes undesirable. To get round this, use:

```
110 a=(a+1)MOD 10:
```

Secondly, it may be necessary for the variable to toggle between 2 values only:

```
210 IF a=2 THEN a=7 ELSE a=2
```

This can be simplified using the formala: a= <sum of values> - <current value>

```
210 a=(7+2)-a
```

Bank switching

This next item deals with the extra memory of a 6128 (or 464 with a memory expansion). The command **OUT &7F00,&C4** (or &C5,&C6,&C7) will load the designated RAM bank into the memory area &4000 to &7FFF in place of the normal memory. The Bank will remain installed, allowing data to be LOADed or SAVEd there until the machine is reset, or **OUT &7F00,&C0** is used to return the memory to normal. Note that no other values should be used.

Disc drive tips

And now here are a whole host of disc drive tips. On the 6128 there is the DERR (disc error) function but unfortunately, this is not available on the 464. As yet, I have not found out if the value can be ascertained somehow, but here are the codes for a 6128, and what they mean:

DERR Number	What it means
144	Bad command (eg. **LOAD "123456789ABCD"** or Disc Missing)
145	File already exists (eg. when RENaming a file)
146	File not found
147	Directory full (more than 64 entries)
148	Disc full
149	Disc changed (eg. while input/output files are open)
150	File is read only (eg. set by CPM)
194	Disc is write protected

Although there are more, these are the common ones.Another thing which is frustrating is the 'Retry, Ignore or Cancel' message which pops up when you've forgotten to insert a disc or it's write protected. On both a 464 and 6128 you can overcome this by typing: **POKE &BE78,&FF**

POKE &BE66,1 reduces the number of times the mechanism grinds away trying to make sense of an unformatted or faulty disc. Sixteen is the normal value but zero equates to 256 - so don't try it as it will take forever.

For 6128 owners, **PRINT PEEK(&AD91)** is the equivalent of **PRINT DERR**.

Protected BASIC

And finally back to BASIC again. To prevent your program from being listed when it has been RUN, type **POKE &AE2C,1** on a 6128 (or **POKE &AE45,1** on a 464) as the first line of your program.

Alternatvely, make sure the first line of your program is **10 REM**
Now type **POKE 372,225**. It will be impossible to list the program, and it can be executed only by using **RUN 20.**

That's it for this issue but, hopefully, we will have found some more odds and ends by the time Issue Nine comes round. Remember if you've any snippets of information, please send them in.

VITAL READING ON M/CODE

012 &BB24 KM GET JOYSTICK

ACTION: This reads the present state of any joysticks attached

ENTRY: No entry conditions

EXIT: H and A contain the state of joystick 0, L the state of joystick 1; all others preserved

NOTES: The joystick states are bit significant and are as follows:

Bit 0 - Up	Bit 1 - Down	Bit 2 - Left	Bit 3 - Right
Bit 4 - Fire2	Bit 5 - Fire1	Bit 6 - Spare	Bit 7 - Always 0

013 &BB27 KM SET TRANSLATE

ACTION: Set the token/character that is assigned to a key when neither SHIFT nor CTRL are pressed

ENTRY: A contains the key number; B contains the new token/character

EXIT: AF and HL corrupted; all others preserved

NOTES: Special values for B are as follows

&80 - &9F	this correspond to the expansion tokens
&FD	causes the CAPS LOCK to toggle on and off
&FE	causes the SHIFT LOCK to toggle on and off
&FF	causes this key to be ignored

014 &BB2A KM GET TRANSLATE

ACTION: Find out what token/character will be assigned to a key when neither SHIFT nor CTRL are pressed

ENTRY: A contains the key number

EXIT: A contains the current token/character that is assigned; HL and flags are corrupted; all others are preserved

NOTES: See above (&BB27) for special values that can be returned

015 &BB2D KM SET SHIFT

ACTION: Set the token/character that is assigned to a key with SHIFT

ENTRY: A contains a key number; B contains the new token/character

EXIT: AF and HL corrupted; all others preserved

NOTES: See above (&BB27) for special values that can be set

016 &BB30 KM GET SHIFT

ACTION: Find out what token/character will be assigned to a key with SHIFT

ENTRY: A contains the key number

EXIT: A contains the current token/character that is assigned; HL and flags are corrupted; all others are preserved

NOTES: See above (&BB27) for special values that can be returned

017 &BB33 KM SET CONTROL

ACTION: Set the token/character that is assigned to a key with CTRL

ENTRY: A contains a key number; B contains the new token/character

EXIT: AF and HL corrupted; all others preserved

NOTES: See above (&BB27) for special values that can be set

018 &BB36 KM GET CONTROL

ACTION: Find out what token/character will be assigned to a key with CTRL

ENTRY: A contains the key number

EXIT: A contains the current token/character that is assigned; HL and flags are corrupted; all others are preserved

NOTES: See above (&BB27) for special values that can be returned

019 &BB39 KM SET REPEAT
ACTION: Set whether a key may repeat
ENTRY: A contains key number; B = &00 then no repeat; B = &FF then repeat
EXIT: AF,BC and HL corrupted; all others presered

020 &BB3C KM GET REPEAT
ACTION: Finds out whether a key is set to repeat or not
ENTRY: A contains a key number
EXIT: If zero is false then it repeats; if zero is true then it does not

021 &BB3F KM SET DELAY
ACTION: Set the time before the first repeat and the repeat speed
ENTRY: H contains the time before the first repeat; L holds the speed
EXIT: AF corrupt; all others preserved
NOTES: The figures in H and L are given in 1/50 seconds; a value of 0 counts as 256

022 &BB42 KM GET DELAY
ACTION: Find out the time before the first repeat and the repeat speed
ENTRY: No entry conditions
EXIT: H contains the time before the first repeat; L holds the speed
NOTES: See above (&BB3F)

023 &BB45 KM ARM BREAKS
ACTION: Arms the break mechanism
ENTRY: DE holds the address of the BREAK handling routine; C contains the ROM select address for this routine
EXIT: AF,BC,DE,HL are corrupt; all others preserved

024 &BB48 KM DISARM BREAK
ACTION: Disarms the break mechanism
ENTRY: No entry conditions
EXIT: AF and HL corrupted; all others preserved

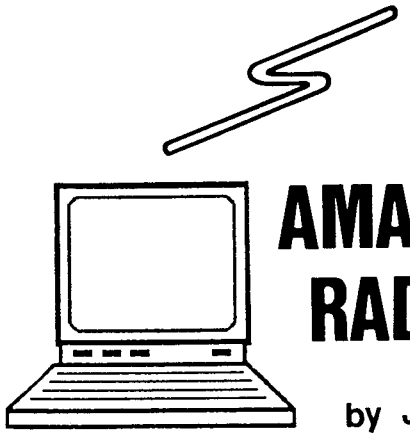
025 &BB4B KM BREAK EVENT
ACTION: Generates a BREAK interrupt if a BREAK routine has been specified
ENTRY: No entry conditions
EXIT: AF and HL corrupted; all others preserved
NOTES: Only takes place if BREAK routine has been set up by KM ARM BREAKS

026 &BB4E TXT INITIALISE
ACTION: Completely initialises the text mode (as when switched on)
ENTRY: No entry conditions
EXIT: AF,BC,DE and HL are corrupted; all others are preserved

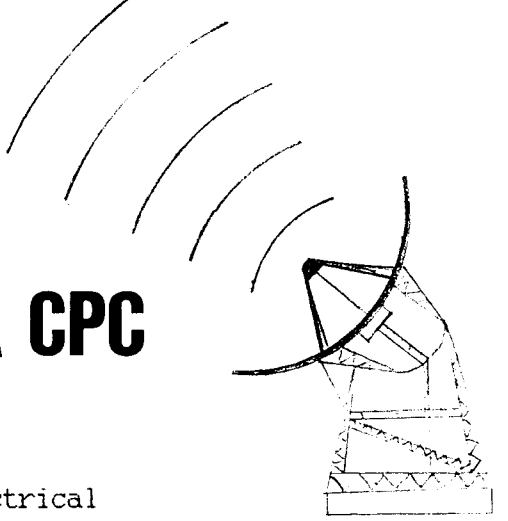
027 &BB51 TXT RESET
ACTION: Resets the control code table and text indirections
ENTRY: No entry conditions
EXIT: AF,BC,DE and HL are corrupted; all others are preserved

028 &BB54 TXT VDU ENABLE
ACTION: Allows characters to be printed to the current stream on the screen
ENTRY: No entry conditions
EXIT: AF are corrupted; all others are preserved

029 &BB57 TXT VDU DISABLE
ACTION: Prevents characters to be printed to the current stream
ENTRY: No entry conditions
EXIT: AF are corrupted; all others are preserved



AMATEUR RADIO ON A CPC



by Jim Proctor

Having been interested in most things electrical from an early age & then being employed in tele-communications, first as an operator & later involved with the installation and maintenance of radio equipment, it was perhaps inevitable that I would progress to amateur radio.

Similarly, with the advent of micro-computers, the urge to find out how to operate them and what made them tick was immediately there, and it was not long before I'd acquired a second-hand CPC. Early efforts with the computer involved a lot of reading & getting to grips with simple BASIC, and I soon realised that here was a tool which had great potential for being used in conjunction with my present hobby of amateur radio. Data storage came first.

Residing on disc, I've a list of all my radio contacts with other amateurs and this is so easily accessed when necessary instead of having to leaf through the written log. I discovered that there was a wealth of public domain software easily obtainable which dealt with electronic design of radio equipment, aerial parameters and propagation, satellite & geographical conditions and other asso-ciated subjects applicable to amateur radio, depending on the particular inter-est of the moment.

All of this was of course, using the computer in perhaps its primary mode, the storage, retrieval and manipulation of data at high speed, thereby reducing drudgery and avoiding error - given that the operator is wide awake and all the bugs have been eliminated! There was however a new interest emerging which inv-olved truly marrying the computer to the radio so that one was the extension of the other - namely data communication.

The simplest approach was, in amateur parlance, RTTY.The letters stand for 'Radio TeleType', whereby teleprinters are joined by a radio link rather than a land line. However despite their electro-mechanical ingenuity, teleprinters are large, noisy and cumbersome and the computer was their natural successor.

With the appropriate software and a relatively simple interface it was not long before I was 'talking' to another amateur via the keyboard. I have regular contacts world-wide now and, as the international language is English, there is no problem in that respect; but if you wish to practise a foreign language it's an ideal opportunity and a hard copy can be obtained on the printer if desired.

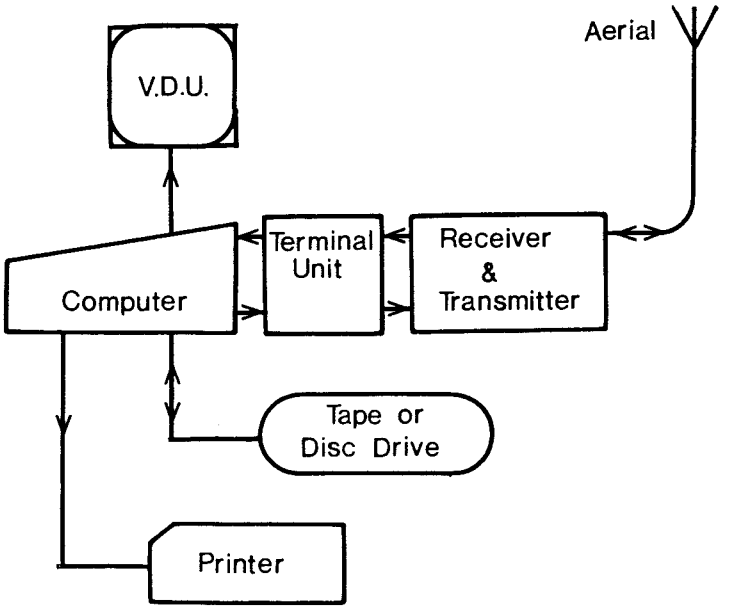
As it is based on an old and, by modern standards, unsophisticated system, 100 percent error free copy isn't always possible especially with a far distant station, due to the vagaries of low power radio communication. However, therein lies the challenge; a continual effort to improve the equipment to reduce error.

Another aspect is listening to commercial broadcasts which use RTTY, especially the Press Associations when you can get the news first-hand! The relevant software is open to experiment and can be tailored to meet the needs of the user so that there is a steady source of interest ideally combining the two hobbies.

Many people enjoy listening to radio transmissions over the whole spectrum & as such require no licence or special qualifications.It's essential to have a good quality communications type receiver for the full benefit to be obtained & this is certainly required for the reception of data systems via the computer.

To be able to transmit on the frequencies allocated to Amateur Radio, a licence is required. To qualify for this, a written examination must be passed covering basic electrical/radio theory and practice & licensing conditions. The examination is in the form of multiple choice questions, and is not as difficult as it may sound; but nevertheless requires a short course of study. For anyone who is interested an initial approach to the Radio Society of Great Britain, see below, is recommended.

The transmitter/receiver is a specialised unit for amateur radio and the terminal unit was built by myself from details supplied to me by the British Amateur Radio Teledata Group. I should mention that terminal units (interfaces) can be purchased ready made, with all the necessary cables, & that they come in many shapes and guises, giving different facilities. The software used is by Scarab Systems & is one of two programs which I hold applicable to the Amstrad CPC 464.



Information on all aspects of Amateur Radio can be obtained from the Radio Society of Great Britain, Lambda House, Cranborne Road, Potters Bar, Herts EN6 3JE.They publish a wide selection of books & would be only too pleased to assist anyone wishing to become involved in the hobby.

There is the British Amateur Radio Teledata Group and membership enquiries should be addressed to Anne Reynolds, G6ZTF, 169 Bell Green Road, Coventry CV6 7GW. As the name suggests they cover the data communication side & are a valuable source for software and hardware.

Another useful group is Sinclair Amstrad Radio User Group, whose secretary is Paul Newman G4INP,3 Red House Lane,Leiston.Suffolk IP16 4JZ. This group deals with all aspects of radio/computing and with particular reference to Sinclair and Amstrad computers.

BY Bob Taylor

BASIC Tokens (2)

!MATCH - AN RSX UTILITY FOR BASIC

Following our look at Tokens in the last issue of PRINT-OUT, this month I'm providing an RSX called '!MATCH' which exploits that information. The command is intended to be used for finding occurrences of statements in a BASIC program.

!MATCH will list all of the lines which contain the required data and this could be anything from a line number or a variable up to a whole BASIC statement. It's possible to restrict the search to a limited range of line numbers if required.

There have been other similar RSXs in the past that have been able to find just variables; they have done so by looking for the correct variable token and then checking the characters of the variable name. They have been restricted to just variables because the multiplicity of tokens would necessitate a different kind of search for each type of constituent of a program line (variable,string, number,command,print item,etc).

A stumbling block, when entering a piece of BASIC as the data to be looked for, is the way that RSX parameters are treated when encountered. Any variables are presented as either the '**two byte value**' or as the '**string descriptor**' (the latter also applies to strings); numbers are converted to a two byte form only; commands would be treated as variables, and print items would probably throw up a **'Syntax Error'**. And this is not how the same 'type' would appear as part of a program line.

On the other hand, this RSX uses the fact that any thing put in a separate statement is automatically converted to the correct tokens etc. on entering the line. So we close the !MATCH command with a colon as if we had finished it, but then we follow this immediately with the statement (or part of statement) which we wish to match. It is this statement, now converted to tokens, which is used as the list of bytes to be searched for.

However our RSX routine will have to manipulate BASIC:

1) to enable this statement to be used as the search list
2) so that the commands in the statement are not actually run.

I mentioned the **BASIC Parser** in the last article - it is the mechanism used by the BASIC interpreter to scan along a program line and carry out the commands it finds there. In order to keep track of where it had reached while it performs an RSX or CALL routine, the BASIC Parser position's address is stored in 2 bytes in the Operating Systems' upper workspace (at **&AE58/9** in the 6128 and at **&AE75/6** in the 464). On completion of the routines the address is taken from these bytes and so the Parser moves on to the next part of the program correctly. However it is possible for us to use and alter this address if we want to.

On entry into an RSX routine the Parser is pointing to the **end of statement byte** (&01) or the **end of line byte** (&00) which immediately follows the statement containing the RSX command.

In our case this position is just before the statement we are going to use to search for, as explained above, so we have a ready made start point provided for the byte list. If we've moved the Parser on to the end of statement byte or the end of line byte following this statement, then when we leave the RSX, this statement will have been skipped over without being run, and the Parser will be none the wiser. Another useful result of moving the Parser on like this is that the incomplete statements used for matching (e.g. 'LEN') will not throw up any '**Syntax Error**'s because they are not seen by the Parser; the processes of token conversion on input, and of Parsing at run time, are carried out to 2 different standards.

HOW DOES IT WORK ?

The method used for searching the program was not the usual one of scanning each line for an occurrence of the first byte in the search list, using the CPIR instruction, as this could throw up spurious sightings, such as finding a string within the characters of an RSX or variable name.

Instead, each token or item in the line is checked against the start of the search list and if not matching then the correct number of bytes is skipped over (by using the information gained last issue) to arrive at the next token or item and the process repeated.

On finding an initial match, then a byte for byte check is made between the search list and the contents of the line from that point.If the line ends before the list has all been checked or if there is a mismatch, then the initial search is resumed either at the token or item following this initial match, or if need be, on the next line.

HOW DO YOU USE IT ?

The **BASIC loader** will alter the routine for whichever version of CPC it is installed in. Once SAVEd as a binary file (done by the program if desired), the routine can be installed elsewhere in memory and it will automatically relocate itself when first CALLed.

The listing of line number occurrences can be paused by using **ESC** once. If used a second time it will exit - whereas any other key will allow the printing to continue.

!MATCH takes two optional parameters. The first is the search Start program line number which defaults to the start of the program if omitted; the line number does not have to be exact (unlike normal BASIC) and if the line specified is not present then the first one following will be used for the start.

The second parameter is the search End program line number which defaults to the end of the program; likewise if the specified line number doesn't exist then the highest one below it will be the last one searched.
The only combinations allowed are :

1) no parameters (whole program search)
2) no End line (search from the start line given, to the end of the program)
3) both parameters (search between the start and end lines inclusively)

Two error messages are present:

1) 'Start line too high' if the Start line is higher than the last line of the program or if higher than the End line parameter.
2) 'Check Parameters' if the following search statement or its preceding ':' are missing.

The full syntax for the command is:

```
!MATCH [,Start Line [,End Line]]: Statement or part statement to be matched
                                  (where [ ] indicate optional parameters)
```

LIMITATIONS

Because of the way that the tokenising routine treats inverted commas, the number and position of these must be correct - each odd numbered quotation mark will open a string and each even one will close it.

1) To search for a string's contents will only require one set of quotes
   eg. :"Hello" will turn up all occurrences of 'Hello' whether by itself or as part of a larger string
2) To look for a string with it's quotes will require either double inverted commas (making three sets in all) or the string will need to be embedded in other tokens
   eg. :"""Hello""" or :PRINT "Hello""" or :PRINT "Hello" CHR$(7) will each look for 'Hello' as a string with quotes, together with the other token(s) etc we have included

Case of letters is important in strings but not for commands or variable names.
Flexibility is provided to allow for equivalence of Variable types, ie.
$ with DEFSTR; ! with DEFREAL and undefined; % with DEFINT and undefined
Numbers must be treated in their entirety. Thus it's not possible to search for all numbers from 100 to 199 by just using the digit 1; every possible value would have to be searched for separately I'm afraid with no wildcards.
Line numbers are treated differently from other numbers when they are tokenised, thus they must always be preceded by a command. GOTO 1000 will work with no problem, but if we want to look for all occurrences of Line Number 1000, it must be forced to become a line number. Evidently GOTO 1000 will still only look for GOTO 1000 and will ignore GOSUB 1000 or RESTORE 1000. However, the !MATCH routine has been written to treat EDIT 1000 as just Line Number 1000 and will then find GOTO 1000, GOSUB 1000, RESTORE 1000, etc - the EDIT part of the statement will be ignored. EDIT by itself can still be searched for, although it should not normally be present in a program.

LISTING

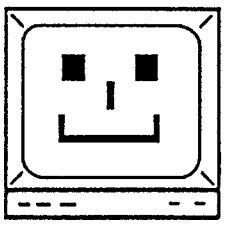
Type in the following Program and SAVE it before RUNning:

```
[F1] 10 'MATCH Loader  by Bob Taylor  (copyright 1990)
[88] 20 MEMORY HIMEM-&2A8:RESTORE 110:PRINT:PRINT"Please wait a few seconds
[B3] 30 FOR lin=0 TO &2A8/8-1:total=0:FOR n=1 TO 8:READ a$
[54] 40 byte=VAL("&"+a$):POKE HIMEM+lin*8+n,byte
[4B] 50 total=total+byte:NEXT n
[0D] 60 READ a$:IF VAL("&"+a$)<>total THEN PRINT:PRINT"Error in line"lin*10+
        110:PRINT:END
[55] 70 NEXT lin:IF PEEK(6)=&80 THEN POKE HIMEM+&93,&75:POKE HIMEM+&F2,&75:POKE
        HIMEM+&30,&93:POKE HIMEM+&31,&CA
[1E] 80 PRINT:PRINT"All M/C loaded":PRINT:PRINT"Press S to save M/C as
        MATCH.BIN":PRINT"or any other key to continue":WHILE INKEY$="":WEND:IF
        INKEY(60)<>-1 THEN a=HIMEM+1:SAVE "MATCH.BIN",B,a,&2A8
[58] 90 PRINT:PRINT"To Load and Initialise !MATCH RSX with a program present
        just Enter:":PRINT"MEMORY HIMEM-&2A8:a=HIMEM+1:LOAD"CHR$(34)"MATCH.BIN"
        CHR$(34)",a:CALL a":PRINT"in Direct Command Mode with the Disc or Tape
        inserted at the correct place"
[EA] 100 END
[8C] 110 DATA 21,34,00,19,06,06,7E,23,11B
[67] 120 DATA E5,66,6F,19,7E,83,77,23,36E
[A9] 130 DATA 7E,8A,77,E1,23,10,EF,44,3C6
[6F] 140 DATA 4D,EB,36,C9,23,C3,D1,BC,4AA
[5D] 150 DATA 21,8F,02,7E,23,CD,5A,BB,335
[FB] 160 DATA B7,20,F8,4F,3E,02,21,55,2D4
[49] 170 DATA CB,C3,1B,00,21,00,76,00,240
[17] 180 DATA 5F,01,A9,00,40,00,75,01,1BF
[75] 190 DATA 78,02,FE,03,30,2F,21,FF,2FA
[E2] 200 DATA FF,EB,21,70,01,FE,01,38,3B3
[E4] 210 DATA 38,D5,DD,56,01,DD,5E,00,37C
[96] 220 DATA 28,01,13,DD,23,DD,23,F5,331
[FB] 230 DATA E5,4E,23,46,78,B1,28,12,2FF
[CF] 240 DATA 23,7E,23,66,6F,ED,52,E1,3B9
[4C] 250 DATA 30,0A,09,18,EB,21,7E,02,1E7
[FD] 260 DATA 18,A9,E1,37,C1,D1,78,3D,420
[E7] 270 DATA 20,C7,38,9C,ED,52,30,98,3C2
[8F] 280 DATA 19,ED,53,06,BF,22,02,BF,301
[E1] 290 DATA ED,5B,58,AE,1A,3D,20,DD,3A2
[8A] 300 DATA 13,1A,FE,96,20,09,13,1A,217
[36] 310 DATA FE,20,13,28,02,1B,1B,E5,276
[CF] 320 DATA CD,5B,02,38,C8,E1,ED,53,44B
[C4] 330 DATA 00,BF,F5,2B,23,F1,CD,09,3C9
[10] 340 DATA BB,FE,FC,20,0C,CD,09,BB,472
[19] 350 DATA 38,FB,CD,06,BB,FE,FC,28,4E3
[6E] 360 DATA 09,ED,4B,06,BF,B7,ED,42,3EC
[71] 370 DATA 38,5F,2A,00,BF,3E,22,BE,29E
[54] 380 DATA 28,05,2B,BE,28,01,23,22,184
[5C] 390 DATA 00,BF,47,18,58,F1,F1,3A,392
[16] 400 DATA 04,BF,B7,28,88,ED,52,ED,456
[6F] 410 DATA 53,58,AE,C8,E1,E5,5E,23,468
[B3] 420 DATA 56,EB,1E,20,01,F6,FF,C5,43A
[F4] 430 DATA 0E,9C,C5,01,18,FC,C5,01,34A
[B2] 440 DATA F0,D8,C5,C1,3E,FF,3C,09,4D0
[1A] 450 DATA 38,FC,ED,42,B7,28,02,1E,362
[37] 460 DATA 30,83,CD,5A,BB,CB,49,28,3D1
[C6] 470 DATA EA,7D,C6,30,06,04,CD,5A,38E
[42] 480 DATA BB,3E,20,10,F9,18,1A,18,26C
[DD] 490 DATA 83,09,4E,23,46,78,B1,28,294
[7B] 500 DATA 99,23,E5,23,23,22,02,BF,2CA
[D7] 510 DATA 0B,0B,0B,0B,0B,ED,43,08,16F
[5A] 520 DATA BF,2A,02,BF,ED,4B,08,BF,3A9
[19] 530 DATA AF,32,05,BF,ED,5B,00,BF,3AC
[91] 540 DATA AF,32,04,BF,D5,E5,CD,5B,486
[70] 550 DATA 02,38,82,E1,D1,78,3C,20,342
[3D] 560 DATA 06,2B,0C,20,FC,18,C0,3D,26E
```

Linechecker

A PROGRAM TYPING AID

All programs in Print-Out have Linecheck codes which are enclosed in brackets at the start of a line. Don't enter them in as they're designed to be used with Linechecker to eliminate errors when typing in programs which appear in this magazine. Please note, all programs will run whether Linechecker is being used or not. For information on how to use Linechecker, please see Issue Three.



```
[7E] 570 DATA B1,28,BC,EB,CD,69,02,EB,4A3
[6F] 580 DATA 28,38,38,2E,7E,FE,22,3A,29E
[92] 590 DATA 05,BF,20,06,2F,32,05,BF,20F
[FA] 600 DATA 18,20,B7,7E,20,1C,FE,0E,2B5
[51] 610 DATA 38,72,FE,19,38,14,28,1A,24F
[46] 620 DATA FE,1D,28,47,FE,7C,28,12,33E
[AC] 630 DATA FE,FF,28,0E,FE,1F,38,0E,396
[AB] 640 DATA 28,10,3E,01,18,13,18,A4,15E
[55] 650 DATA 18,A7,3E,02,18,0A,3E,03,162
[39] 660 DATA 18,05,3E,06,0B,0B,0B,0B,08D
[40] 670 DATA 0B,0B,C5,47,1A,BE,13,23,230
[19] 680 DATA 20,14,10,F8,C1,3A,04,BF,2FA
[DE] 690 DATA B7,20,89,3C,ED,43,08,BF,393
[A1] 700 DATA 22,02,BF,18,D3,23,10,FD,2FE
[CC] 710 DATA C1,18,CB,1A,FE,1E,20,CE,3C8
[7E] 720 DATA 13,23,0B,7E,23,E5,66,6F,29C
[F5] 730 DATA 23,23,23,1A,BE,13,23,0B,182
[61] 740 DATA 20,02,1A,BE,13,0B,E1,23,21C
[5F] 750 DATA 28,CB,18,AA,FE,0C,28,18,2FF
[4E] 760 DATA 30,1A,FE,0B,28,09,FE,03,285
[C0] 770 DATA 28,0E,38,03,1A,18,16,1A,0D3
[54] 780 DATA FE,0D,28,17,C6,02,18,0D,237
[C2] 790 DATA 1A,3C,18,09,1A,FE,02,28,1B9
[4F] 800 DATA 0A,FE,0B,28,06,FE,04,28,26B
[B4] 810 DATA 02,FE,0D,23,0B,23,0B,23,18C
[7A] 820 DATA 0B,20,18,13,13,13,1A,E6,17C
[19] 830 DATA DF,BE,28,05,F6,20,BE,20,3BE
[40] 840 DATA 0A,13,23,0B,CB,7F,28,EE,2AB
[28] 850 DATA AF,18,AD,CB,7E,23,0B,28,313
[A2] 860 DATA FA,18,A7,1A,FE,22,20,0A,31D
[B8] 870 DATA 23,13,1A,FE,22,20,03,23,1B6
[73] 880 DATA 13,1A,FE,01,C0,13,1A,1B,234
[53] 890 DATA FE,C0,C8,FE,97,C8,37,C9,5E3
[73] 900 DATA 4D,41,54,43,C8,00,43,68,298
[8E] 910 DATA 65,63,6B,20,50,61,72,61,2D7
[C0] 920 DATA 6D,65,74,65,72,73,00,53,2E3
[91] 930 DATA 74,61,72,74,20,4C,69,6E,2FE
[C1] 940 DATA 65,20,74,6F,6F,20,68,69,2C8
[59] 950 DATA 67,68,00,00,00,00,00,00,0CF
```

In the next article in the series I hope to explain the way that the Variables, Arrays and Strings areas are used in BASIC.

Small Ads

DATA PD LIBRARY has around 2 Megabytes of PD software on tape and disc. For the latest newsletter and full list, send a SAE or 30p. Or send for the Starter Pack - over 20 programs to try out, send a SAE,tape/disc and 50p. Cheque/PO payable to T. Kingsmill at 202 Park Street Lane,Park Street,St Albans,Herts AL2 2AQ.

PLAY MATES the CPC Games Fanzine, full of reviews, tips and pokes. Now carrying Bonzo Meddler news also. Order issues 4, 5 and 6 for £2.50 from Carl Surry, 37 Fairfield Way, Barnet, Herts EN5 2BQ.

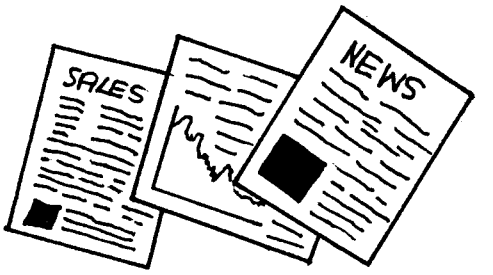
WANTED - Soft 968 Firmware Manual to buy - full price offered or costs refunded for a short loan. Anyone need a 464 manual (some pages in a funny order) ? Write to: R Bignell, 24 Deangarden Rise, High Wycombe, Bucks HP11 1RE.

FOR SALE - Amstrad CPC 464 Instruction Manual, as new, £8 + £1 post/packing, or will exchange for four 3" discs. P.H. Breckin, 161 Longsight Road, Holcombe Brook, Bury, Lancs BL8 4DA. 'Phone 020488-3443

WANTED - To buy or on loan, Amstrad Action magazines Numbers 18 & 23, plus Xmas 1985, Feb 1987 & Dec 1987. P.H.Breckin - address and phone number as above.

FOR SALE - 2 Homebrew programs, Casino Blackjack (a realistic simulation of the gambling game) and Wordsearch (a utility for solving wordsearch puzzles) As reviewed in Print-Out Issue Six. The programs cost £4.50 together, and this includes the cost of a CF-2 disc. Contact Barrie Snell, 19 Rochester Road, Southsea, Portsmouth PO4 9BA.

News



Christmas Sales

Once again, the CPC news scene seems devoted to the Plus computers which many national magazine journalists had tipped as the most popular machine for first-time buyers this Christmas.

Unfortunately, the figures don't quite support this. It would appear that the Plus computers, whilst selling in average quantities, did nothing to dent sales of the 16-bit giants or the aging Commodore 64. It isn't all bad news though, as most retailers reckon that the Pluses' time will come during the next year; even though it looks likely that Amstrad will soon release a new 16-bit PC-Compatible games machine which could take the bottom out of the 8-bit market.

Likewise, the GX-4000 console did not shift in any significant number and did not even attempt to rival the likes of Nintendo and Sega. However it wasn't just Amstrad who suffered, as Commodore's console also failed to sell over Christmas. It was also a dismal time for Amstrad's other 8-bit, the Spectrum, and it seems increasingly likely that its days are numbered.

Amstrad isn't greatly worried by these facts, adding to the speculation that the company has something impressive up its sleeve.

Compatibility?

On the same theme, confusion seems to abound concerning compatibility of both hardware and software. with Amstrad saying one thing, and independent retailers contradicting them. Most of the inadequacies of the 464 Plus and 6128 Plus seem to have been sorted out by WAVE - though at the cost of your warranty - allowing a disc drive to be used on a 464 Plus, and loading from tape on a 6128 Plus. But Amstrad have seemed very reticent on these modifications and are unwilling to be drawn on anything.

Cheaper PD

Public Domain for the CPC is a thriving business and we can bring you details of Tony Kingsmill's DATA PD. The copying charge has now been reduced from 2p per Kilobyte to 1p per K. Also the starter pack has been updated to include some of the more recent programs. For more information, contact Tony at 202 Park Street Lane, Park Street, St Albans, Herts AL2 2AQ.

New Articles

Finally, we would like to say that many new series are in the pipeline at the moment, as a result of letters which we have received. Therefore, we would like to remind all of our readers that we are always open to suggestions and ideas & all are considered very carefully.

SOUND

An introduction to the CPC's sound chip

Part One

Finally, we have managed to put together the promised series on the various SOUND commands. In it, we'll be looking at all aspects of 'noise' on the CPC and this will include special effects, music, and possibly even speech synthesis.

In the last issue, the SOUND command was introduced in a simple form and we looked at the various numbers (known as parameters) which can follow it. However the SOUND command can be far more complicated than it appeared last time and, to produce realistic music and sound effects, we'll need to look at its assosciated commands - namely the Tone and Volume Envelopes.

You may remember that last time I said that a note's volume on a 464 ranged from 0 to 7 and on a 6128, from 0 to 15. Now, although this is confusing enough, there is another problem. With a volume envelope attached the volume ranges from 0 to 15 on both computers!

However, the actual loudness at the maximum value is the same on both a 464 and a 6128 - the 'levels' of volume are just split up differently. And as we are dealing with volume envelopes this issue, there shouldn't be any discrepancies.

Volume Envelopes

For this issue's section, we will be working on the note that's produced by the command **SOUND 1,400,150,15** If you type it in and listen to it, you will hear a sound of pitch 400, which lasts for 1.5 seconds at maximum volume. However, it is noticeable how it remains at a constant volume. But in reality, no sound ever stays at the same volume for its entire duration, and to simulate any changes on the CPC, we have got to use the ENV (or volume envelope) command.

To hear ENV in action, enter the following line and compare it with before:

```
ENV 1,5,3,30:SOUND 1,400,150,0,1
```

You should now hear the note gradually rising from silence to its maximum volume in the same time of 1.5 seconds. Before looking at the ENV command, we must look at the slightly altered SOUND expression. There's now a ,1 put on the end of the SOUND command and this identifies which of the fifteen available envelopes is to be used for that particular note. Also the volume has been set to 0, and this is now the start volume on which the envelope acts.

Note that this volume envelope does not have to be redefined every time you want to play a note, but is instead stored in the computer's memory, ready to be used at anytime. You can check this by just typing in: **SOUND 1,400,150,0,1**

So what is a volume envelope? Firstly, it is very important to realise that the ENV command produces no sound on its own, but only defines a volume envelope which can be used by the SOUND command at a later date. To prove this, using the same envelope as before, retype **ENV 1,5,3,30** and you should hear nothing. But if you now enter **SOUND 1,400,150,0,1** you should hear the note gradually increasing.

But how do we get those numbers which follow the ENV command? If you listen to the note again (type **SOUND 1,400,150,0,1**), you should notice five changes (or STEPS) in the volume.The note's volume increases in steps of three and each step lasts for 0.3 seconds. Thus the simplest form of the ENV command has the syntax:

ENV N,P,Q,R

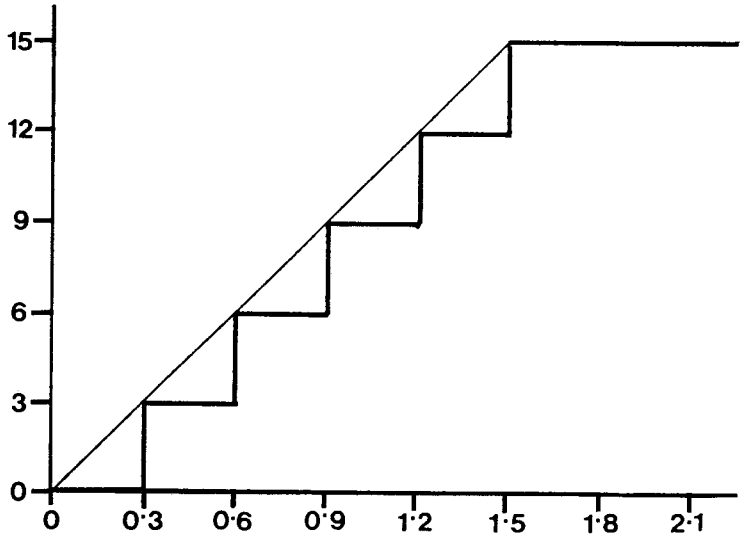
N is a number between 1 and 15 and identifies which envelope is to be defined. This number is also attached to the end of the SOUND command to tell the CPC which envelope it is meant to use.

P is the number of steps there are. There can be upto 127 of these steps in an envelope, but in our example above, there are five steps.

Q tells the computer how much it is to increase or decrease the volume in each step. In our envelope the volume was to increase by three in each step. This parameter can have any value between -128 and 127.

R informs the computer of how long each step is to last.Like the duration of a note, it's measured in hundredths of a second and can be any value between 0 and 255. In the envelope above it is 30, which equates to 0.3 seconds.

Now that we know what the numbers mean, all we've got to do is to try and decide what values to put in our ENV command. The best way of doing this is to draw out what we want the sound to be like, in terms of volume (see Figure I). Next we've got to divide it up into a number of steps (P) and finally measure how long each step lasts (R) and the volume increase or decrease (Q).



Using the envelope **ENV 1,5,3,30**: This causes the note to increase to its maximum volume in the 1.5 seconds that it is playing. This diagram shows why.

There are five steps, each lasts 30/100ths of a second (ie 0.3 of a second) and with each step the volume increases by three volume 'levels'. As a result the volume goes from 0 (silence) to 15 (its maximum) in 1.5 seconds. However the increases are not smooth but are jerky and distinct. It is important to have this in mind when designing sound effects and sound envelopes.

Listed here are some more envelopes which you might like to try. Each of them can be heard by entering:

```
SOUND 1,400,150,0,n
```

where 'n' is the required envelope number.

```
ENV 2,15,1,10
ENV 3,2,6,75
```

Here are a few which have decreasing volumes. These must be started off with:

```
SOUND 1,400,150,15,n
```

And these envelopes are:

```
ENV 4,15,-1,10
ENV 5,5,-3,30
```

AY-3-8912 sound chip

The CPCs all use the AY-3-8912 sound chip (as does the Atari ST) and, although designed over 10 years ago, this sound chip is capable of producing quite respectable 3-channel music and sound effects - in fact it was first used in coin-op arcade machines. However the Plus computers, despite using the same chip, include some major improvements to the sound quality. Firstly the monitors have built-in stereo speakers and this provides amplification & clarity The 'custom chip' at the heart of the Pluses extra capabilities plays the tunes & effects without any intervention from the processor, thus allowing the games to be played even faster.

At the start of this article. I said that a note's volume must be between 0 and 15, but this is not strictly true. All of the examples given upto this point have kept the volume between these two figures, but what happens if we let it go outside these? Try it and see using: `ENV 6,30,1,5:SOUND 1,400,150,0,6`

The volume increases to its maximum loudness of 15 and then, because the CPC's a cunning beast, it loops back to 0 and then continues to increase the volume from there!! It also works the other way: `ENV 7,30,-1,5:SOUND 1,400,150,15,7`

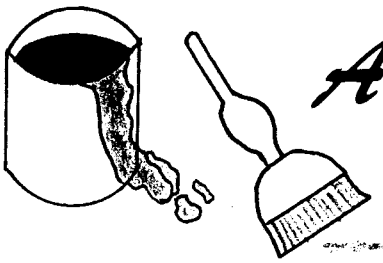
This can be useful when designing sound effects. For example, the envelope below could be used as a machine gun firing: `ENV 8,75,4,2:SOUND 1,400,150,0,8`

In all of the examples given so far, the duration of the sound has been 1.5 seconds. Similarly, the envelopes have all been designed to last for 1.5 seconds - you can see how long they are meant to go on for, by multiplying the number of steps (P) by the length of each step (Q).

Of course, you can make the sound play for a longer or shorter time than is required by the envelope by altering the duration in the SOUND command, and this can produce some interesting results.

If you set the duration in the SOUND command to zero, however, the envelope then controls the length of the note on its own. Thus, this cures the problem to a certain extent - but more on it in a later issue.

We have now looked at making sounds get either louder or quieter in detail. However, in real life, sounds don't just increase or decrease in volume; they do both. With our simple ENV command, it is not possible to mimic this and so there is a far more complicated form of ENV available to us. In Issue Nine, we will be looking at this, and also studying some ways of making music on the CPC!!



ARTIST DESIGNER

MODIFICATIONS

Over the past few months we've received so many letters about the Artist Designer program that we have dedicated these two pages to various modifications that can be used to improve it.

Here then are your suggestions, problems and our replies:

I typed it in without Linechecker and I inevitably made a few typing errors, most of which I have sorted out. There remained one or two 'bugs' which may not appear on the tape/disc. However, this is what I found:

1) On saving a picture to tape/disc, the filename was corrupted. On introducing the line **25 OPENOUT "d":MEMORY HIMEM-1:CLOSEOUT** this cured the problem. I then had to change line 2520 to **2520 IF my>381 AND my<399 THEN RUN 30** to avoid having to keep a disc in the drive on 'ERASE PICTURE'.
2) LOAD/SAVE menu was staying up on 'CHANGE FILENAME' and 'EXIT MENU'. Line 340 needed to be changed to **340 IF mopt=1 THEN 350 ELSE CALL &900C**

R. BIGNELL, HIGH WYCOMBE

PRINT-OUT: On my 6128, the first problem did not occur, but I think that it may be something to do with a bug in the 464's Operating System when used with a disc drive. Still if any reader has this problem, the above correction would seem to do the trick. The second bug, however, was our fault and somehow the line got corrupted. This even got onto some of the early program tapes/discs. Printed below are some modifications which allow the cursor to be speeded up; to move the cursor at 5 times its normal speed, press CTRL and a cursor key.

```
[A4] 1590 IF INKEY(8)=0 AND x>9 THEN x=x-2
[45] 1595 IF INKEY(8)<>128 THEN 1600 ELSE IF x>19 THEN x=x-10 ELSE IF x>9 THEN
          x=x-2
[12] 1600 IF INKEY(1)=0 AND x<620 THEN x=x+2
[4B] 1605 IF INKEY(1)<>128 THEN 1610 ELSE IF x<610 THEN x=x+10 ELSE IF x<620
          THEN x=x+2
[F1] 1610 IF INKEY(0)=0 AND y<391 THEN y=y+2
[13] 1615 IF INKEY(0)<>128 THEN 1620 ELSE IF y<381 THEN y=y+10 ELSE IF y<391
          THEN y=y+2
[A4] 1620 IF INKEY(2)=0 AND y>30 THEN y=y-2
[87] 1625 IF INKEY(2)<>128 THEN 1630 ELSE IF y>40 THEN y=y-10 ELSE IF y>30
          THEN y=y-2
[84] 3880 IF INKEY(8)=0 AND x>9 THEN x=x-1
[7B] 3885 IF INKEY(8)<>128 THEN 3890 ELSE IF x>19 THEN x=x-10 ELSE IF x>9 THEN
          x=x-1
[0C] 3890 IF INKEY(1)=0 AND x<620 THEN x=x+1
[8B] 3895 IF INKEY(1)<>128 THEN 3900 ELSE IF x<610 THEN x=x+10 ELSE IF x<620
          THEN x=x+1
[B4] 3900 IF INKEY(0)=0 AND y<381 THEN y=y+1
[C5] 3905 IF INKEY(0)<>128 THEN 3910 ELSE IF y<371 THEN y=y+10 ELSE IF y<381
          THEN y=y+1
[83] 3910 IF INKEY(2)=0 AND y>30 THEN y=y-1
[AD] 3915 IF INKEY(2)<>128 THEN 3920 ELSE IF y>40 THEN y=y-10 ELSE IF y>30
          THEN y=y-1
```

SCREEN DUMP

Many people have written in requesting a Screen Dump routine for use with the Artist Designer program in Issue 6 of Print-Out. The Screen Dump described here is intended for MERGEing with 'Artist' only. It will give an inverted print-out (black Ink on white Paper) of the work area across a sheet of A4 paper.

Type in and SAVE the BASIC listing below. (It is not a complete program, and will do very little by itself). LOAD the original Artist program, and then MERGE this new routine with it. When the main program is RUN, it will install the new Machine Code routine ready for your Screen dumps.

To perform a dump any time that the cursor is present on the work area, just use **CONTROL and D together**. To stop a dump before it is completed, hold down the **DEL Key** until the Printer stops.

For future use, once the new lines have been proved to be correct just repeat the LOAD and MERGE sequence and then SAVE the resulting program.

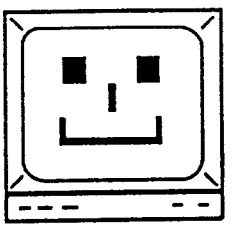
Due to the geometry of the DMP 2000 Printer, circles will be flattened when they are printed with an aspect ratio of 6:5. This is due to the spacing of the pins on the print head, and also seems to afflict the STAR LC10 Colour Printer (and, I suspect quite a few others).

The only solution to this problem would result in a much larger print-out (30 times bigger), requiring many sheets of paper to be cut and joined together, and I feel that this is somewhat excessive for the requirements of the program.

Another problem with the DMP 2000 is that it cannot tolerate continuous horizontal lines of graphics - it will miss out every other dot. The result of this is that both the Complete Fill and Dense Speckle fills appear almost identical, so any designs prepared for dumping need to have this taken into consideration.

```
[F8] 1655 IF INKEY(61)=128 THEN TAG:MOVE ox,oy:PRINT CHR$(232);:CALL &9249:MOVE
          ox,oy:PRINT CHR$(232);:GOTO 1590
[F1] 3940 'PATTERN FILL and SCREEN DUMP Loader by Bob Taylor for PRINT-OUT
[6A] 3960 FOR lin=0 TO &2A8/8-1:total=0:FOR n=0 TO 7:READ a$
[44] 4681 DATA 00,00,00,00,00,00,00,00,000
[DB] 4682 DATA 00,CD,2E,BD,30,09,CD,09,2C7
[7D] 4683 DATA BB,FE,7F,28,74,18,F2,3E,41C
[93] 4684 DATA 1B,CD,D4,92,3E,40,CD,D4,46D
[74] 4685 DATA 92,3E,1B,CD,D4,92,3E,41,39D
[DC] 4686 DATA CD,D4,92,3E,06,CD,D4,92,4AA
[7F] 4687 DATA 06,1E,21,7E,01,11,01,00,0D6
[41] 4688 DATA C5,E5,01,03,7E,3E,1B,CD,352
[64] 4689 DATA D4,92,3E,59,CD,D4,92,78,4A8
[30] 4690 DATA CD,D4,92,3E,02,CD,D4,92,4A6
[7A] 4691 DATA E1,E5,3E,02,C5,D5,E5,F5,57A
[AC] 4692 DATA CD,F0,BB,B7,C1,78,28,01,491
[B5] 4693 DATA 37,E1,D1,2B,2B,17,30,ED,373
[F7] 4694 DATA CD,D4,92,13,C1,10,E1,0D,405
[9F] 4695 DATA 20,DE,23,23,3E,0A,CD,D4,32D
[EE] 4696 DATA 92,3E,0D,CD,D4,92,F1,C1,4C2
[28] 4697 DATA CD,09,BB,FE,7F,28,02,10,348
[74] 4698 DATA AC,3E,1B,CD,2B,BD,3E,40,338
[E7] 4699 DATA CD,2B,BD,C9,CD,2B,BD,D8,50B
[9A] 4700 DATA CD,09,BB,FE,7F,20,F5,F1,514
[4E] 4701 DATA 18,E7,00,00,00,00,00,00,0FF
```

All programs in Print-Out have Linecheck codes which are enclosed in brackets at the start of a line. Don't enter them in as they're designed to be used with Linechecker to eliminate errors when typing in programs which appear in this magazine. Please note, all programs will run whether Linechecker is being used or not. For information on how to use Linechecker, please see Issue Three.



ADVANCED MODE

An intro to RSXs *part 4*

Using and Programming ROMs

by **BOB TAYLOR**

In the first three articles in this series, we have covered Initialisation, Error Handling, passing and retrieving Parameters and finally Relocation. Until now, the RSXs we were considering were intended to be loaded into RAM. However, RSXs can also be installed in ROMs so providing permanent access to their extra commands; in fact, any RAM resident RSX could be put into ROM form.

WHY PUT RSXS ONTO ROMS?

The most obvious advantage is that no time need be spent LOADing & initialising the RSX routine – it is instantly available for use. Another reason is that the space which had to be reserved for the routine in RAM is no longer required, so allowing more room for the BASIC program and its variables and strings.

Are there any disadvantages? Of course! The main stumbling block for most people is the extra hardware needed to place the routine into a ROM, and then to run it there; I refer to the ROM Blower and ROM board respectively. Once installed in a ROM another drawback will eventually become apparent: if any bugs are discovered in the code, or if the routine needs to be enhanced to meet changing situations, the routine cannot be altered.

A further minor drawback is that the routine is accessed slightly slower than it was when in RAM. The reason for this is that the Operating System, on encountering an external command (as all extra commands preceded by the "|" character are called) looks first at any RAM RSXs for a match and, on not finding one, it then looks through each ROM in turn, starting with that at the lowest Select address; therefore, it will take that much longer before it is matched and can be used. This extra delay isn't large but should be borne in mind if the external command is one which requires fast servicing or which occurs many times within a loop in BASIC.

Let us suppose that these drawbacks do not detract from the positive advantages of ready access and increased free memory, so...

HOW DO WE WRITE FOR A ROM?

The ROMs which we are referring to are meant to be installed as sideways ROMs (see later). Given the correct hardware, a ROM Board, it is possible to fit ROMs in any of 252 positions, each position being given a single byte address – called its ROM Select address.

Background ROMs (the type for which RSXs should be written) can be placed anywhere in Select addresses &00 to &0F on the 6128 (&01 to &07 for the 464). Above these addresses, only Foreground ROMs can be fitted, and then only at the first available position; they can also be fitted randomly in the lower addresses just as the Background ones can.

Firstly the routine itself will usually be identical for both medium. Apart from the fact that the Stack Pointer will be moved down 8 bytes from its start position instead of 6 in the case of a RAM RSX, every point mentioned in the previous articles relating to parameters passed, and use of registers still applies - and as it's not normal to need to access the stack, we can thus discount that slight difference.
Secondly the initialisation routine is not required. Instead, the command name & the jump address are included in the existing name table and command tables present in the ROM.The command table takes exactly the same form as it did with RAM based RSXs: a two byte address of the start of the list of names of all commands present in the ROM, followed by a list of JumP instructions to the routines servicing those commands. Note that all commands must be present in this one list - a second list is not allowed. However there's no limit to the number of commands which a ROM may have in its one command table - the limit finally comes down to how many routines can be squeezed into the 16K available in a full sized ROM.
Thirdly, of course, there is no need for a re-location routine to alter absolute addresses; this work will have been done during the writing and proving stage.

WRITING COMMAND ROUTINES

Due to the permanence of ROM routines, a complete & thorough testing of the routine in RAM is essential. It's fortunate then that RSXs in RAM or ROM require identical routines, so enabling testing to be carried out in RAM before transfer to ROM. It is also helpful that RSX routines can be CALLed instead of needing to be initialised as RSXs.Both of these conspire to allow the routine to be written for any convenient area of RAM during most of its stages.
An Assembler is most useful for such work and if the routine is lengthy, will be indispensable in 're-directing' the code to its final position, once development has been completed. The need for this re-direction is as follows. The RAM available for writing the code in runs from &0176 to &ABFF at most (although standard practice is to use the block &4000 to &7FFF because this matches the final destination for the code with an simple offset of &8000). The ROM on the other hand will be located at &C000 up to &FFFF.
Once it has been tested, some sort of re-location will be needed for the routine to run in ROM. Modified re-location routines, of the types suggested in previous articles might do for this purpose, but would probably become quite unwieldy if large numbers of addresses have to be changed. This is where Assemblers come in. Most have a facility for storing code at one place in memory while being able to modify addresses as if it had actually been assembled at another area of memory, and only at the last assembling would this have to be used.

An alternative to this method of working is to use what is called sideways RAM. The extra ROMs which are plugged into ROMboards at the back of our CPCs are known as sideways ROMs; the reason for this is that if the memory map of our computers is drawn out, such ROMs will be positioned alongside the BASIC ROM which is itself alongside the upper 16K of RAM - usually used for Screen Memory. Sideways RAM is RAM connected in place of one of these sideways ROMs. Being RAM it's alterable (with suitable software) & being in place of a ROM, it will be run by the operating system as if it actually were a ROM. I use a proprietary version of this system called an AMRAM; it's no longer available unfortunately, nor is anything identical made by anyone else. However, there is a device called a RAMROM, produced by Microstyle, which has two 16k sideways RAMs switchable to different ROM Select positions. One disadvantage it has compared to the Amram is that it requires to be re-loaded with code each time the computer is switched on (mine has a battery to maintain the data at all times). Using such a device, it's possible to write code at the location it is intended to run.

The first few bytes of a ROM need to contain a certain sequence of values:

```
&C000  byte of &01      This signifies that the ROM is a background ROM.
&C001  byte giving ROM mark No                  (user selectable)
&C002   "      "    ROM version No                  "       "
&C003   "      "    ROM modification level          "       "
&C004  2 byte address of start of list of Command Names
&C006  JumP instruction to ROM initialisation routine
&C009   "       "       "  first command's handling routine
&C00C   "       "       "  next command's routine
 etc
```

There are several other possibilities for the ROM type 'byte' at &C000, although not ones that we can use in our context:

&00 for foreground ROM. This type of ROM use also allows external commands to be present in a Command Table. However once invoked, control is taken from the calling program and doesn't return to it - not much use in BASIC

&80 special type of foreground ROM that is reserved for the resident operating language of the computer - in our case BASIC

&02 for extension ROMs. These are used in a very long foreground program which cannot be contained in one 16K ROM, extra ROMs containing the rest of this program can be placed at the next 3 ROM select addresses & given this type byte. This allows faster access from one of these ROMs to another by using RST 2 instructions

NOTE The Mark, Version and Modification bytes are completely at the disposal of the user - I usually use them for the date instead, as their 'proper' use still requires extra written information including the date.

The list of Command Names pointed to by the address at &C004 is mostly identical to that for RAM RSXs: each letter is in capitals with the last character of each name having bit 7 set,the last name is followed by a byte of &00. The difference is that the first name of the ROM list is for the ROM initialisation routine and in standard practice, this should not be available to be invoked from BASIC. To this end it is normal to include a Space within the name and this will make it impossible to call. However, it is a rule which could be broken. If you should happen to write an initialisation routine which is also useful from BASIC, there is no reason why a Space should be included.

The main purpose for the initialisation routine is to allow it to reserve working space in RAM. On entry, which is from the Operating System only, DE and HL contain respectively the lower and upper limits of free RAM. Working space is reserved by moving the lower limit upwards or the upper limit downwards (or even both), by altering the values of DE or HL on exit. What use is made of any space reserved is entirely up to the user - however on entry later to any RSX routine in that ROM, IY will hold the address that was contained in HL on exiting the initialisation routine (allowing simple indexing to up to 127 bytes of reserved memory).

Other uses for the initialisation routine include setting up any routine variables (perhaps in the reserved memory area), and printing any messages. On returning to the Operating System from the initialisation routine, the Carry flag must be set.

The alternative ways of using the JumP block discussed in previous articles could also be applied to our ROM. However in general, as the ROM space is not at such a premium, I always use the JumP block in the conventional way.

Routines running inside a ROM perform more or less as they did when in RAM. However, depending upon whether the ROM board causes 'Wait' states to enable it to use slower EPROM chips, the actual routine may run slower.

Writing to memory above &BFFF affects the Screen memory, even though a ROM, occupying the same range of addresses, may be enabled. But any 'READ' operations (with the ROM enabled) would be made from the ROM and not the underlying Screen RAM. To eliminate this problem, should a Screen 'READ' be required, a CALL is needed to a routine in RAM which will temporarily switch off the Upper ROM, read from the Screen RAM, switch on the Upper ROM again and RETurn to it. Fortunately for us, such a routine is already available in the Firmware in the form of RST 4 - RAM LAM. This will read one byte at a time from the address held in HL.

Access to other ROM's routines which also occupy the same area of memory as our ROM, is also catered for in the Firmware with the RST 3 FAR CALL. This operates very much like the CALL to &001B we used with Error Handling in the first article in this series, but with the advantages that the C and HL registers are left free for passing more data. This is done by placing the routine address and ROM Select No (which HL and C held with the &001B CALL) in a three byte data block called a Far Address. The RST 3 instruction itself is followed by the address of this data block and a CALL into the other ROM's routine is performed by the Operating System, temporarily switching off our ROM.

Similarly, the Lower ROM is available to the Upper ROM, either directly by enabling the Lower ROM, and retrieving data from it or CALLing a routine in it, or by the indirect facilities provided by RST 1 or RST 5. These both perform a JumP operation and not a CALL. For this reason, they cannot be included in a sequence of code, but need to be situated separately and CALLed from within the sequence - it is essential that the routine being used in the Lower ROM will terminate with a RET instruction to return control back to our calling ROM.

A feature of these Firmware RSTs is that all switching of ROMs is carried out by the Firmware as is the restoration of the original various ROM states after the RST has been performed. Use of the central 32K of RAM by the ROM for data or routines, does not require any use of RSTs as there is no need to alter any ROM selection.

USING RSXS FROM M/CODE

The location of an RSX routine can be ascertained by making the HL register point to a copy of the RSX name in RAM, and then calling the Firmware's KL FIND COMMAND at &BCD4.On exit from this CALL, HL will point to the start of the routine (via its JumP block entry), and C will hold the Select Address of the ROM containing the routine if it is in ROM (the programmer will need to know if the routine is in RAM or ROM). The Carry flag will be set if the RSX is found. If it is, the routine address and Select Address can be installed in a Far Address in RAM for a RST 3 to access an RSX in ROM.

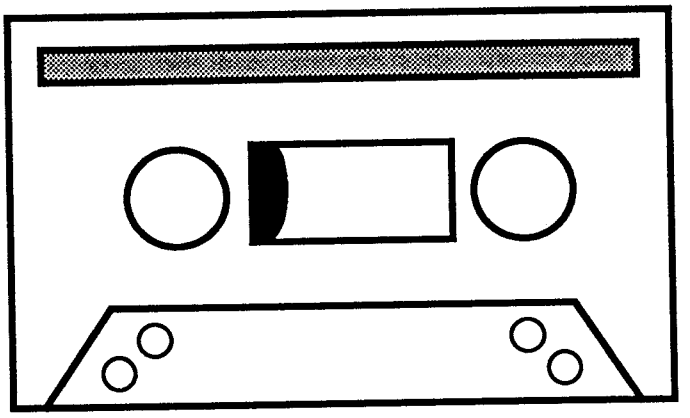
An RSX in RAM is easier to use, although not just with a 'JP (HL)' as this would leave no Return address on the stack. Rather, a CALL is needed to any location conveniently having the JP (HL) instruction (eg. CALL &001E).

If the RSX requires no parameters, its use is quite straightforward. RSXs with parameters can be run in the same way, but the expected parameters must be assembled in a form usable to the RSX. They need to be arranged as a block of 2 byte values, with low byte before high byte, and with the last parameter at the lowest address and the first at the highest. This parameter block can be in RAM, or in the same ROM as the calling routine. However it should not be part of the machine stack as it is in BASIC. The IX register should point to the low byte of the last (lowest) parameter & the A register must hold the number of parameters in the block. The values of the parameters should be what the routine expects of course, in accordance with the information given earlier in this series.

RSXs in RAM or ROM are really intended to be accessed from BASIC and return to BASIC when serviced. Usually, they extend the range of functions available to BASIC, either to make up for deficiencies or to create exotic extra facilities.

Sometimes however they can provide non-BASIC utilities like Word Processors or Monitors which seem to take over the whole machine while they are running. Which ever use you find for RSXs, the provision for extra commands has certainly enhanced our computers and I hope that this series of articles may have prompted you to think about writing your own RSXs.

Offers



Please make all cheques payable to Print-Out but any postal orders should be made out to T J Defoe as this saves the Post Office a great deal of time and effort. Unless it cannot be avoided, it is advisable not to send cash through the post.

ISSUE NINE

If you wish to order a copy of Issue Nine in advance, you may do so by sending a cheque / postal order for £1.10 to the usual address. With luck we will have it published by about 30th February and it will be forwarded to you as soon as it's available. Program tapes and discs are also available in advance.

PROGRAM TAPES/DISCS

We supply both program tapes and discs for ALL issues of Print-Out (including back issues). The prices also include a booklet to explain how the programs work plus postage and packing. The cost for each of the program cassettes is:-

a) A blank tape (at least 15 minutes) and 50p (p+p)
or **b)** £1.00 (which also includes the price of a tape)

And the cost for each of the program discs is :-

a) A blank formatted disc and 50p (p+p)
or **b)** £3.00 (which also includes the cost of a MAXELL/AMSOFT disc) *

BACK COPIES

We've still a supply of ALL back issues of Print-Out available and the price is £1.10 which includes postage and packing. Alternatively you can order both a back issue and its corresponding tape or disc by sending:-

a) £1.75 - includes the tape, the required issue and postage and packing.
or **b)** £3.75 - includes the disc (genuine MAXELL/AMSOFT disc) * and also the required issue and postage.

* When ordering using this particular method please allow about **21 days** for delivery as we must rely on outside suppliers for the discs.
* Please also note that one side of one CF-2 disc will hold all the programs from upto six issues. Therefore, the cost is £3.00 for a disc plus one set of programs and then 50p for each additional issue thereafter.

There are two forms of subscription to Print-Out available and they are:-

a) Three issues - approximately half a year

b) Six issues - approximately a full year

Although we do try and produce one magazine every two months this is not always possible due to other outside engagements and therefore exact release dates are not given in the magazine. Because of this, we are unable to guarantee that six issues will be produced in a year, or three issues in half a year. However, for a year's subscription you will be sent six issues no matter when they are published, and the same applies to a half-yearly subscription. If we stop producing the magazine, we promise to refund the cost of all unmailed issues.

The prices for subscriptions to Print-Out are as follows :-

NO OF ISSUES	UNITED KINGDOM	EUROPE	REST OF THE WORLD
SINGLE	£1.10	£1.50	£2.00
THREE ISSUES	£3.30	£4.50	£6.00
SIX ISSUES	£6.60	£9.00	£12.00

Send to: **Print-Out,8 Maze Green Road,Bishop's Stortford,Herts.**

NAME (block capitals please) .....

ADDRESS .....,

.....,

.....,

POSTCODE .....

Please send me the following items :-

DESCRIPTION	ISSUE NUMBER	QUANTITY	PRICE EACH	PRICE
			TOTAL PRICE	

I enclose a cheque/postal order/cash to the value of £..... Please make all cheques payable to PRINT-OUT & make postal orders out to T. Defoe. Thank you.

NB: Please include any details of a subscription to Print-Out in the above order form. Please write the issue you wish your subscription to start with in the ISSUE NUMBER column, and the length of the subscription in the quantity column.